Name: Luyando Kwenda
Class: ENM 5020
Due Date: 11 April 2023
Assignment: 3

Implicit Euler Method Application to Lotka Volterra Equations

Contents

1 Introduction

Lotka Volterra Equations are a set of nonlinear differential equations that model the predator-prey relationship in an ecosystem. We assume no migration into and out of the system hence forming a closed loop. The terms a,b,c and d are constant parameters whilst p and q are modifiers that represent competition.

$$\frac{dx}{dt} = (a - by)x - px^2 \tag{1}$$

$$\frac{dy}{dt} = (cx - d)y - qy^2 \tag{2}$$

Eq. (1) in this setup will represent the population of the prey species whilst Eq. (2) will represent the population of the predator species. Our aim is to solve this system of equations using Implicit Euler Equations (Backward Euler). Implicit Euler solves approximations using

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}) \tag{3}$$

However, the y_{n+1} appears on both sides of the equations so we need oat to use newtons method given an initial guess until the result converges within a specific tolerance. We further analyse the trends in population of the the two species over a given time and calculate the error that results in using Implicit Euler.

In our initial problem, we are give an equal value of constant parameters and no over (p and q are 0), but we also want to see what happens if vary these variables and include competition.

2 Problem Setup and Formulation

2.1 Critical Points

In order to find the critical points, we set each differential equation to zero and solve any x and y solution as shown below

$$(a - by)x - px^2 = 0 \tag{4}$$

$$(cx - d)y - qy^2 = 0 \tag{5}$$

After solving these equations algebraically, we are able to arrive at 4 possible critical points:

$x = 0$	$x = \frac{a-by}{p}$
$y = 0$	$y = 0$
$y = \frac{-d}{q}$	$y = \frac{ca-dp}{bc+qp}$

In the description of the given problem, we are to set p and q equal to 0 hence we only consider 2 critical points. These two points are derived by setting our new equation (p=q=0) and solving for each variable which gives $(0, 0)$ and $(\frac{d}{c}, \frac{a}{b})$. We then are able to check what kind of critical point using a jacobian matrix i.e

$$J(0,0) = \begin{pmatrix} a - by & -bx \\ cy & cx - d \end{pmatrix}\Bigg|_{(0,0)} = \begin{pmatrix} a & 0 \\ 0 & -d \end{pmatrix} \tag{6}$$

$$J(1,1) = \begin{pmatrix} a - by & -bx \\ cy & cx - d \end{pmatrix}\Bigg|_{(1,1)} = \begin{pmatrix} 0 & -b \\ c & 0 \end{pmatrix} \tag{7}$$

From Eq. (6) we have two eigenvalues $\lambda_1 = a = 1$ and $\lambda_2 = -d = -1$ with each magnitude greater than zero and having opposite signs shows that this is an unstable critical point (saddle point). As for Eq. (7) we get two complex eigenvalues $\lambda = \pm i\sqrt{bc} = \pm i$ which shows that this is a fixed point and in our case is a center of a spiral (physically impossible).

Note, the above calculations are done with a=b=c=d=1 and p=q=0. Otherwise, we would have to find what kind of critical points we listed in the table above for different constants.

2.2 Implicit Euler and Newtons Method

We will consider a first order ode, and the apply a backward difference approximation

$$\frac{dy}{dt} = f(t, y) \tag{8}$$

$$\frac{dy}{dt} = \frac{y_n - y_{n-1}}{h} \tag{9}$$

hence we get

$$\frac{y_n - y_{n-1}}{h} = f(t_n, y_n) \tag{10}$$

we can the rewrite the equation in the following form

$$y_n = y_{n-1} + hf(t_n, y_n) \tag{11}$$

Because we are looking for the next time step, n+1, we can replace n with n+1 and replace n-1 with n and this is how Eq. (3) was derived.

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}) \tag{12}$$

This equation is similar to Explicit Euler except that the explicit case uses new guess approximations from the previous and current result which is known unlike implicit euler where we need the next and current guess to form the new approximation ie

$$y_{n+1} = y_n + hf(t_n, y_n) \tag{13}$$

The main difference between the two methods is that the Explicit Euler method is faster and easier to implement but may be unstable for certain differential equations, while the Implicit Euler method is more stable but requires more computational resources to solve. The choice of which method to use depends on the specific problem being solved and the desired level of accuracy and stability. In the population we are trying to solve the Lotka-Volterra equations can exhibit stability issues when using the Explicit Euler method, especially when the time step is large. This can lead to unrealistic and physically impossible behavior in the solution, such as oscillations or divergence.

In contrast, the Implicit Euler method is unconditionally stable for the Lotka-Volterra equations, meaning that it can handle any time step size without encountering stability issues. This method involves solving a nonlinear equation at each time step, which can be computationally more expensive than the Explicit Euler method, but it typically results in a more accurate and stable solution. Hence its use in this assignment.

To solve for the impicit method, we need to use Newtons method to solve for an initial guess. We also use the jacobian and residual matrices to converge to a solution with R_1 being the first row of the residual and R_2 as the second ie

$$R = \begin{pmatrix} x_{n+1} - hax_{n+1} + bx_{n+1}y_{n+1} + hpx_{n+1}^2 - x_n \\ y_{n+1} + hdy_{n+1} - hcx_{n+1}y_{n+1} + hqy_{n+1}^2 - y_n \end{pmatrix} \tag{14}$$

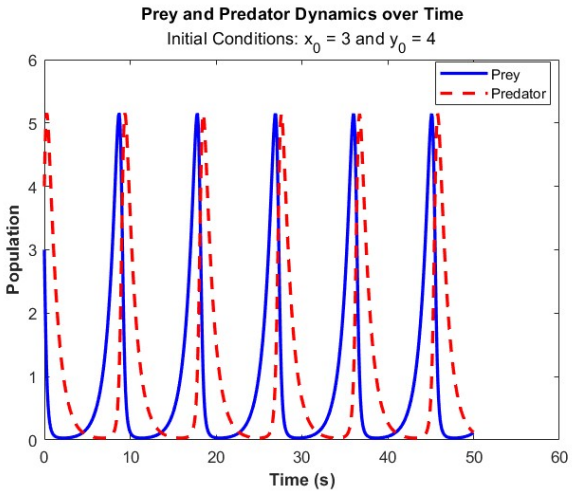
$$J = \begin{pmatrix} \frac{\partial R_1}{\partial x_{n+1}} & \frac{\partial R_1}{\partial y_{n+1}} \\ \frac{\partial R_2}{\partial x_{n+1}} & \frac{\partial R_2}{\partial y_{n+1}} \end{pmatrix} \tag{15}$$

When choosing an initial guess, we choose a random set of numbers that do not include the critical points of the initial equations. We then set $x_{n+1} = x_n$ and plug into the above equations and then solve the linear system of equations $J\delta = -R$. The new delta values are then added to the previous n+1 to form the next next guess until we converge to a solution. The code in the appendix provides full detail on the implementation. We also choose a time step h (0.0001) which is kept constant.

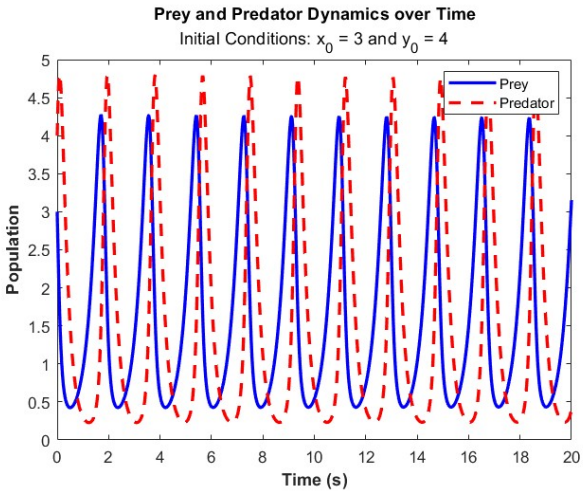
3 Results

3.1 Time Results

Given a specific time step (h=0.0001), for a=b=c=d=1, we notice that in all the graphs, the peaks of the predators and prey are equal. In the case where these constants are not equal, we get a difference in peak height. In the figure 1b, we see that the predator population peaks higher than the prey because the c and d variables were larger. We would be able to get a similar result in the case of the prey if a and b were larger.

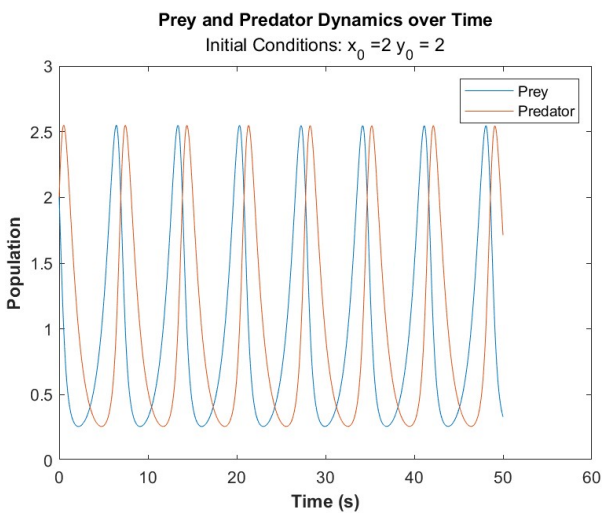


(a) a=b=c=d=1

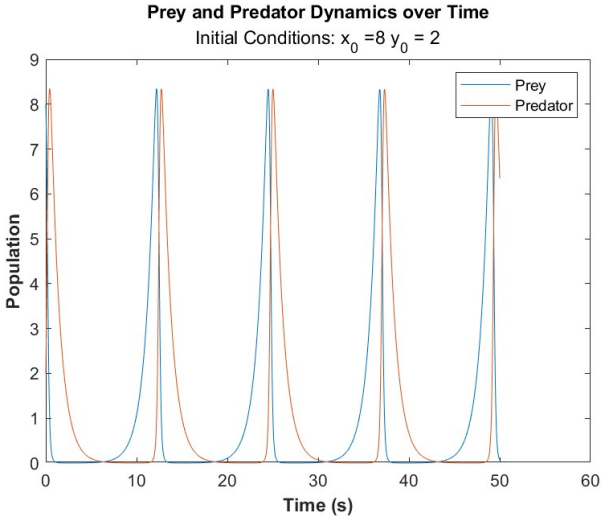


(b) a=3, b=2, c=3, d=5

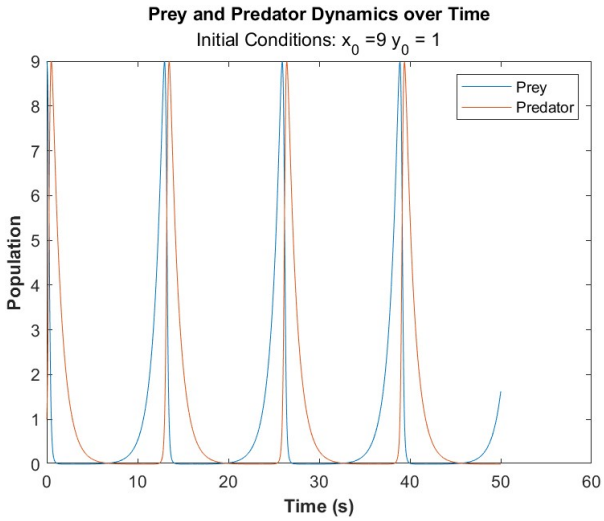
Figure 1: Population Time Plots



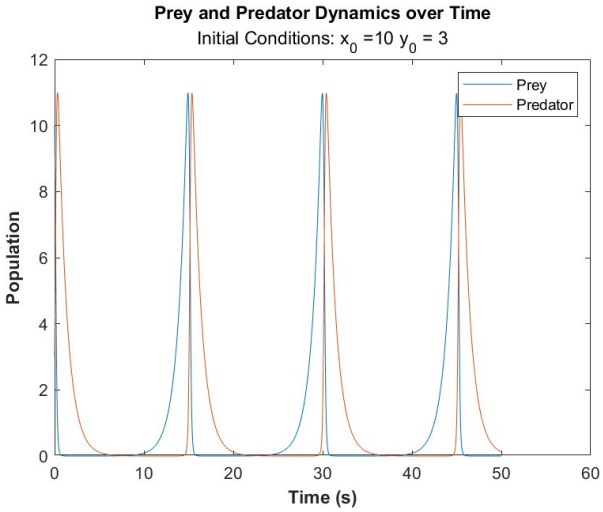
(a)



(b)



(c)



(d)

Figure 2: Population Time Plots with Different Initial Values

3.2 Phase Space Trajectory

Another notable observation is the frequency of the waves produced. From figure 2a, we see that this option had the highest occurrence of peaks and was closest to the critical point. As we move further away from the critical point, the number of cycles reduces from 4 to 3. The initial condition determines how are away we are from the critical point. We also not that if any of the calculations start with and initial condition of (1,1), we get a flat line (y=1) because that is the steady solution and the derivative at that point if zero and no integration can occur.

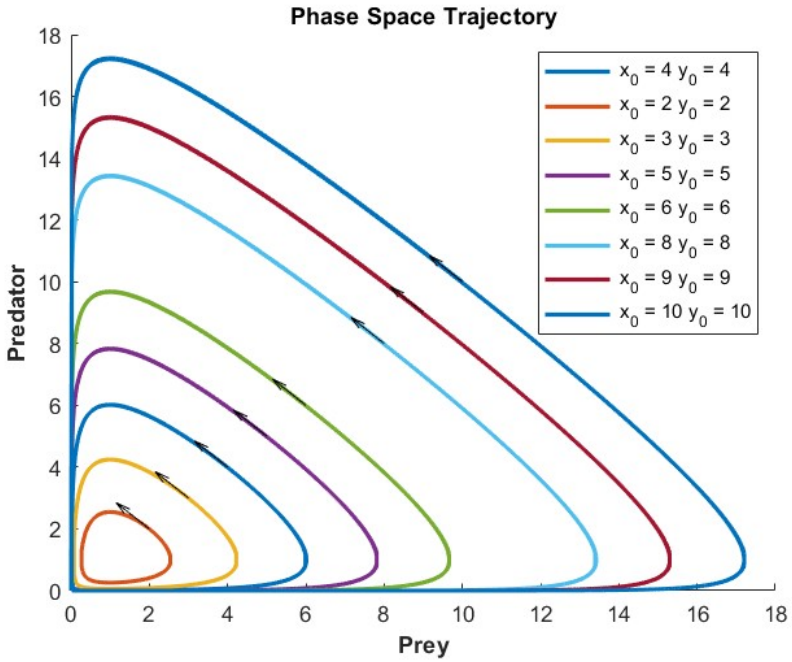
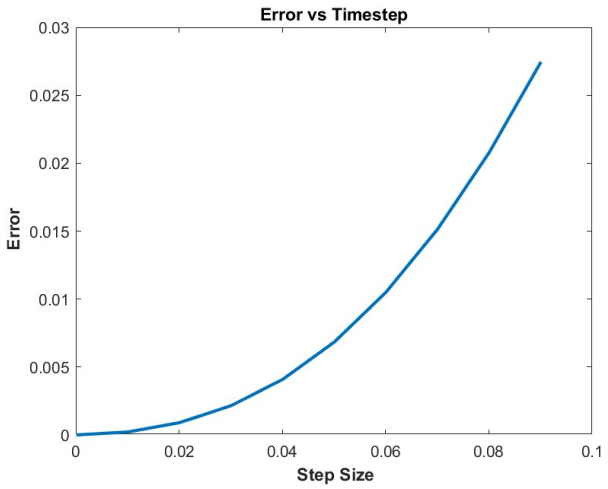


Figure 3: Phase Plot

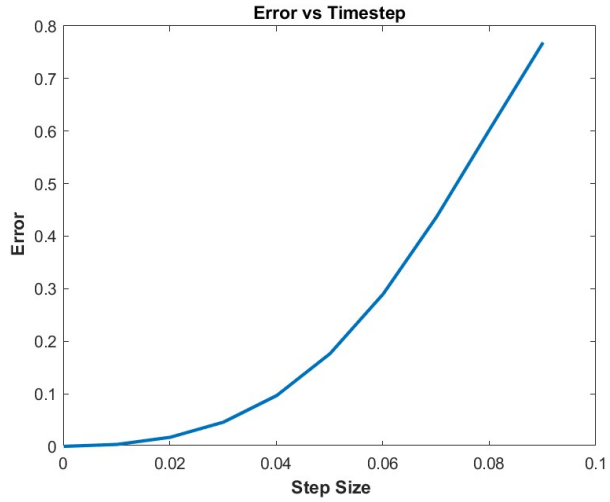
We can also look at the predator-prey relationship using Phase Plots as shown in figure 3. As the prey population grows which is the lower part of the graph closer to the x axis, we see a lower population growth for the predators. Because of an abundance in food, the predator population increases gradually while the prey population decreases. The top of the phase is the peak of the predator population and we can assume that there is insufficient food so we begin to see a sharp decrease in the predator population. This decrease allows for re-population of the the pray and the cycle begins again. This process forms a closed loop and seen and the only difference is whether the initial condition is closer to the stable point. The further we get away from this point, the larger the loop. At the stable point, we do not for a loop but rather it would be represented by a dot.

3.3 Error

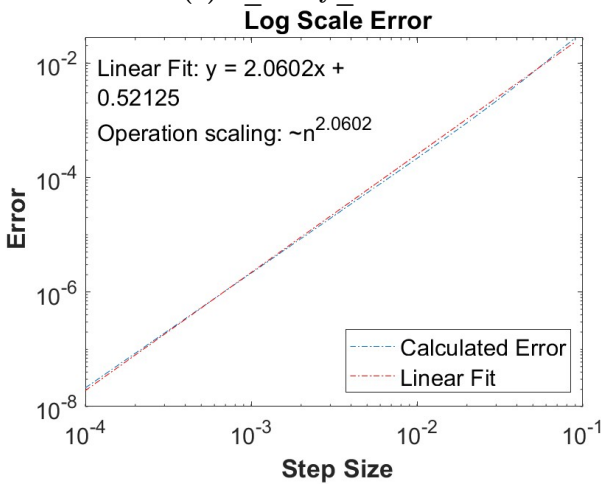
Theoretically, we know that the global error is in $O(h^2)$. In out case, we need to find the error by each time step $\delta t = h$ which means our error is h^2. To prove this, we assume a time step, h, and calculate the x and y values. We then break up that time step into 2 parts making the new time step $h/2$ and use this new value to evaluate the result. This new result will the be the new initial condition and produces an output at the same location as our initial time step. Because we do this twice, the error will be $2(\frac{h}{2})^2 = \frac{h^2}{2}$. We then do this for multiple step sizes to obtain multiple values. From figure 4 we are able to see plots of error vs step size. Figure 4a and 4b show trend of a quadratic error which was our initial assumption. As the step size grows larger, the error grows quadratically. We then further used the log scale to show the operation scaling which produced a value of approximately 2.



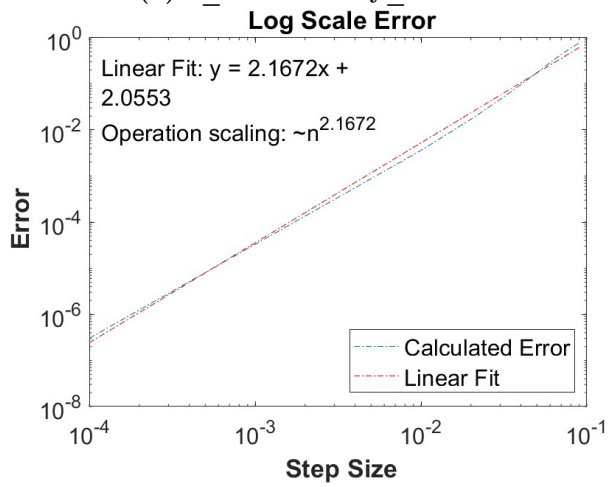
(a) x_0 = y_0 = 3



(b) x_0 = 10 and y_0 = 1



(c) x_0 = y_0 = 3



(d) x_0 = 10 and y_0 = 1

Figure 4: Error Plots

4 Conclusion

In summary, we had been tasked with solving the Lotka Volterra System of equations which in our case was modified. We then determined the stability of the solution and found a general case in which we included and overcrowding parameter. However, we only analyzed our solution in the case where the overcrowding term was equal to zero. We came to a conclusion that we had two critical points, one a saddle point (0,0) and the other was a stable fixed point (1,1). The kind of critical points was determined through the use of a eigenvalues derived from a Jacobian matrix. We then used Implicit Euler Methods to obtain the population patterns over time and analyse the predator-prey relationship and well. The implicit method was chosen over the explicit because of its numerical stability which is a challenge of this system of equations. As we start of further form the critical point, we saw higher periodicity ie fewer peaks. Based on the phase diagram, the relationship displayed a closed loop. Lastly, we looked at the error generated from different time steps. As we increased the time step, we saw a quadratic increase in the error generated. This kind of error growth makes the implicit method more ideal than the explicit method. In all the calculations performed, the error scaled to the power 2.

5 Appendix

Below is the code used in the analysis of the problem

5.1 Main Code

```
1
2  % choose the number of steps we want
3  steps = 500000;
4
5  %select initial conditions
6  xi = [4 2 3 5 6 8 9 10];
7  yi = [4 2 3 5 6 8 9 10];
8
9  f1=[0 0 1];
10 %allocate constants
11 a = 1;
12 b = 1;
13 c = 1;
14 d = 1;
15 p = 0;
16 q = 0;
17
18 %chose a time step dt
19 h = 0.0001;
20
21 %tolerance for convergence
22 tol = 10^-6;
23
24 %Preallocate vectors that store output
25 xvec = zeros(1,steps+1);
26 yvec = zeros(1,steps+1);
27 time = zeros(1,steps+1);
28 figure
29 hold on
30 for jj = 1: length(xi)
31     x0 = xi(jj);
32     y0 = yi(jj);
33
34     xvec(1) = x0;
35     yvec(1) = y0;
36
37     for ii = 1: steps
38         [new_x, new_y] = NewtsMethod(a,b,c,d,p,q,x0,y0,tol,h);
39
40         %store values
41         xvec(ii+1) = new_x;
42         yvec(ii+1) = new_y;
43
44         %assign new x0 and y0 values for next step
45         x0 = new_x;
46         y0 = new_y;
47
48         time(ii+1) = time(ii) + h;
49
```

```
    end

    plot(xvec,yvec,'DisplayName'," x_0 = "+num2str(xi(jj))+ " y_0 = "+num2str(
        ↪ yi(jj)),LineWidth=2)



end


ylabel('Predator','FontWeight', 'bold')
xlabel('Prey','FontWeight', 'bold')
title('Phase Space Trajectory')
legend;

[dx, dy] = points(a,b,c,d,p,q,xi,yi);
quiver(xi,yi,dx,dy,.3,'k','HandleVisibility','off');
```

5.2 Newtons Methods

```
function [new_x, new_y] = NewtsMethod(a,b,c,d,p,q,x0,y0,tol,h)

er = 1;

x = x0;
y = y0;

%initialize jacobian matrix
J = zeros(2,2);

%initialize residual vector
R = zeros(2,1);

while er>tol

    %populate jacobian
    J(1,1) = 1 - h*(a-b*y0) + 2*h*p*x0;
    J(1,2) = h*b*x0;
    J(2,1) = -h*y0*c;
    J(2,2) = 1 - h*(c*x0-d) + 2*h*q*y0;

    %populate residual
    R(1) = x0 - h*(a-b*y0)*x0 + h*p*x0^2 - x;
    R(2) = y0 - h*(c*x0-d)*y0 + h*q*y0^2 - y;

    delta = J \ -R;
    dx = delta(1);
    dy = delta(2);

    %calculate error
    er = sqrt(dx^2 + dy^2);

    %update n+1 values
    new_x = x0+dx;
```

```
    new_y = y0+dy;

    %assign new initial values
    x0 = new_x;
    y0 = new_y;
end



end
```

5.3 Calculate Derivative

```
function [dx,dy] = points(a,b,c,d,p,q,x,y)

dx = (a-b.*y).*x -p*x.^2;
dy = (c.*x -d).*y -q*y.^2;
dx = dx.*0.1;
dy = dy.*0.1;

mags = sqrt(dx.^2+dy.^2);
dx = dx./mags;
dy = dy./mags;

end
```

5.4 Error Script

```
% choose the number of steps we want
steps = 500000;

%select initial conditions
x0 = 3;
y0 = 3;

%allocate constants
a = 1;
b = 1;
c = 1;
d = 1;
p = 0;
q = 0;

%tolerance for convergence
tol = 10^-6;

h_vec = 0.0001:0.01:0.1;

error_vec = zeros(1,length(h_vec));

for ii = 1:length(h_vec)
    h = h_vec(ii);
    [x1, y1] = NewtsMethod(a,b,c,d,p,q,x0,y0,tol,h);
```

```
26
27      %divide the step size
28      h = h/2;
29      [x2, y2] = NewtsMethod(a,b,c,d,p,q,x0,y0,tol,h);
30
31      %plug in new values
32      [x3, y3] = NewtsMethod(a,b,c,d,p,q,x2,y2,tol,h);
33
34      xer = x1-x3;
35      yer = y1-y3;
36
37      err = xer^2 +yer^2;
38      error_vec(ii) = sqrt(err);
39
40  end
41
42
43  figure
44  plot(h_vec,error_vec,'LineWidth',2)
45  xlabel('\bf Step Size')
46  ylabel('\bf Error')
47  title('Error vs Timestep')
48  exportgraphics(gcf,"Actual error" + "x_0 =" + num2str(x0)+ " y_0 = " + num2str(
        ↪ y0) +".jpg")
49
50  p_p1 = polyfit(log10(h_vec),log10(error_vec),  1);
51  myfitp1 = (h_vec.^(p_p1(1))) * 10^p_p1(2);
52  figure
53  loglog(h_vec,error_vec,'-.')
54  hold on
55  loglog(h_vec,myfitp1,'r-.')
56  xlabel('\bf Step Size')
57  ylabel('\bf Error')
58  title('\bf Log Scale Error')
59  if sign(p_p1(2)) == 1
60  a = annotation('textbox',[.14 0.6 .5 .3],'String',{['Linear Fit: y = ', num2str
        ↪ (p_p1(1)), 'x + ' num2str(p_p1(2))],['Operation scaling: ~n^{' num2str(
        ↪ p_p1(1)) '}']},'EdgeColor','none');
61  else
62  a = annotation('textbox',[.14 0.6 .5 .3],'String',{['Linear Fit: y = ' num2str(
        ↪ p_p1(1)) 'x - ' num2str(abs(p_p1(2)))],['Operation scaling: ~n^{' num2str
        ↪ (p_p1(1)) '}']},'EdgeColor','none');
63  end
64  l =legend('Calculated Error','Linear Fit','Location','Southeast');
65  a.FontSize = 15;
66  l.FontSize = 15;
67  set(gca,'fontsize', 15);
68  exportgraphics(gcf,"log error" + "x_0 =" + num2str(x0)+ " y_0 = " + num2str(y0)
        ↪  +".jpg")
```

5.5 Time Analysis

```
1
2   % choose the number of steps we want
```

```
steps = 500000;

%select initial conditions
xi = [3 2 3 5 6 8 9 10];
yi = [4 2 5 4 9 2 1 3];

f1=[0 0 1];
%allocate constants
a = 1;
b = 1;
c = 1;
d = 1;
p = 0;
q = 0;

%chose a time step dt
h = 0.0001;

%tolerance for convergence
tol = 10^-6;

%Preallocate vectors that store output
xvec = zeros(1,steps+1);
yvec = zeros(1,steps+1);
time = zeros(1,steps+1);

for jj = 1: length(xi)
    x0 = xi(jj);
    y0 = yi(jj);

    xvec(1) = x0;
    yvec(1) = y0;

    for ii = 1: steps
        [new_x, new_y] = NewtsMethod(a,b,c,d,p,q,x0,y0,tol,h);

        %store values
        xvec(ii+1) = new_x;
        yvec(ii+1) = new_y;

        %assign new x0 and y0 values for next step
        x0 = new_x;
        y0 = new_y;

        time(ii+1) = time(ii) + h;

    end

figure
plot(time,xvec,time,yvec)
xlabel('Time (s)','FontWeight', 'bold')
ylabel('Population','FontWeight', 'bold')
legend('Prey','Predator')
title('Prey and Predator Dynamics over Time')
subtitle("Initial Conditions: x_0 =" + num2str(xi(jj))+ " y_0 = " + num2str(yi(
    ↪ jj)))
```

```
58 exportgraphics(gcf,"x_0 =" + num2str(xi(jj))+ " y_0 = " + num2str(yi(jj)) +".
       ↪ jpg")
59
60
61 end
62
63 close all
```